VLADEMIRO LANDIM JUNIOR

INTRODUÇÃO A PROGRAMAÇÃO USANDO A LINGUAGEM HARBOUR.

FORTALEZA, CEARÁ

2016

VLADEMIRO LANDIM JUNIOR

INTRODUÇÃO A PROGRAMAÇÃO USANDO A LINGUAGEM HARBOUR.

Uma introdução aos aspectos básicos da programação de computadores usando a linguagem Harbour como ferramenta de aplicação dos conceitos aprendidos.

Área de concentração: Introdução a Programação, Linguagem Harbour, Computação, Informática, Algoritmos.

FORTALEZA, CEARÁ

2016

LANDIM, Vlademiro.
Introdução a programação usando a linguagem Harbour.
/ Vlademiro Landim Junior. 2016.
65p.;il. color. enc.

VLADEMIRO LANDIM JUNIOR

INTRODUÇÃO A PROGRAMAÇÃO USANDO A LINGUAGEM HARBOUR.

Área de concentração: Introdução a Programação, Linguagem Harbour, Computação, Informática, Algoritmos.

Aos meus Pais.

AGRADECIMENTOS

Agradeço a Paulo César Toledo e a todos que participam do fórum Clipper On Line (http://www.pctoledo.com.br/forum). A solidariedade e a atenção de todos vocês foi essencial para a conclusão desse trabalho. **Com certeza não vou poder citar todos** pois muitos me ajudaram mesmo sem saber, e na pressa do momento eu acabei esquecendo o nome da pessoa ou até mesmo nem lendo o nome. Segue abaixo uma lista muito parcial (a ordem não reflete a importância, fui me lembrando e digitando):

- Paulo César Toledo
- José Quintas
- “Maligno”
- Fladimir
- Rochinha
- “Asimoes”
- Claudio Soto
- Vailton
- Pablo Cesar
- janio
- Stanis Luksys
- Jairo Maia
- Itamar M. Lins Jr.
- “RobertoLinux”
- porter
- “alxsts”
- Eolo
- “paiva_dbdc”
- “rbonotto”
- “vagucs”
- “sygecon”

- “Imatech”

Caso alguém encontre algum erro nesse material a responsabilidade é minha. Por favor envie um e-mail com as correções a serem feitas para vlad@altersoft.net com o título “E-book Harbour”.

“Suponham que um de vocês tenha um amigo e que recorra a ele à meia-noite e diga ‘Amigo, empreste-me três pães, porque um amigo meu chegou de viagem, e não tenho nada para lhe oferecer’. E o que estiver dentro responda: ‘Não me incomode. A porta já está fechada, e eu e meus filhos já estamos deitados. Não posso me levantar e lhe dar o que me pede’. Eu lhes digo: Embora ele não se levante para dar-lhe o pão por ser seu amigo, por causa da importunação se levantará e lhe dará tudo o que precisar. Por isso lhes digo: Peçam, e lhes será dado; busquem, e encontrarão; batam, e a porta lhes será aberta”

(Jesus Cristo, Filho do Deus vivo - Evangelho de Lucas 11:5-9)

RESUMO

Esse livro busca ensinar os princípios de programação utilizando a linguagem Harbour. Ele aborda os conceitos básicos de qualquer linguagem de programação, variáveis, tipos de dados, estruturas sequenciais, estruturas de decisão, estruturas de controle de fluxo, tipos de dados complexos, funções, escopo e tempo de vida de variáveis. Como a linguagem utilizada é a linguagem Harbour, o presente estudo busca também apresentar algumas particularidades dessa linguagem como comparação de *strings* e macro substituição. Palavras-chave: Introdução a Programação, Linguagem de programação, Linguagem Harbour, Sistemas de Informação, xBase, Clipper.

ABSTRACT

This book seeks to teach the principles of programming using the language Harbour . It covers the basics of any programming language , variables , data types , sequential structures , decision structures , flow control structures , complex data types , functions, scope and lifetime of variables. As the language is the language Harbour , this study also seeks to present some peculiarities of this language as a comparison textit string and macro replacement.

Keywords: . . .

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 A quem se destina esse livro

Este livro destina-se ao iniciante na programação de computadores que deseja utilizar uma linguagem de programação com eficiência e produtividade. Os conceitos aqui apresentados se aplicam a todas as linguagens de programação, embora a ênfase aqui seja no aprendizado da Linguagem Harbour. Mesmo que essa obra trate de aspectos básicos encontrados em todas as linguagens de programação, ela não abre mão de dicas que podem ser aproveitadas até mesmo por programadores profissionais. O objetivo principal é ensinar a programar da maneira correta, o que pode acabar beneficiando ao programador experiente que não teve acesso a técnicas que tornam o trabalho mais produtivo. Frequentemente ignoramos os detalhes por julgar que já os conhecemos suficientemente.

Esse livro, portanto, foi escrito com o intuito principal de beneficiar a quem nunca programou antes, mas que tem muita vontade de aprender. Programar é uma atividade interessante, rentável e útil, mas ela demanda esforço e longo tempo de aprendizado. Apesar desse tempo longo, cada capítulo vem recheado de exemplos e códigos para que você inicie logo a prática da programação. Só entendemos um problema completamente durante o processo de solução desse mesmo problema, e esse processo só se dá na prática da programação.

Por outro lado, existe um objetivo secundário que pode ser perfeitamente alcançado : nas organizações grandes e pequenas existem um numero grande de sistemas baseados em Harbour e principalmente em Clipper. Isso sem contar as outras linguagens do dialeto *xbase* [1]. Esse livro busca ajudar essas organizações a treinar mão-de-obra nova para a manutenção dos seus sistemas de informações. Se o profissional já possui formação técnica formal em outras linguagens ele pode se beneficiar com a descrição da linguagem através de uma sequencia de aprendizado semelhante a que ele teve na faculdade ou no curso técnico. Dessa forma, esse profissional pode evitar a digitação de alguns códigos e apenas visualizar as diferenças e as semelhanças entre a linguagem Harbour e a linguagem que ele já conhece.

1.2 Pré-requisitos necessários

Como o livro é voltado para o público iniciante, ele não requer conhecimento prévio de linguagem de programação alguma. Porém, algum conhecimento técnico é requerido, por exemplo :

1. familiaridade com o sistema operacional Microsoft Windows: navegação entre pastas, criação de pastas, cópia de conteúdo entre uma pasta e outra, etc.

2. algum conhecimento matemático básico: expressões numéricas, conjunto dos números naturais, inteiros (números negativos), racionais e irracionais. Você não

[1] Flagship, FoxPro, dBase, Sistemas ERPs baseados em *xbase*, XBase++, etc.

precisará fazer cálculo algum, apenas entender os conceitos.

Os conhecimentos desejáveis são :

1. criação de variáveis de ambiente do sistema operacional: para poder alterar o PATH do sistema.
2. conhecimento de comandos do Prompt de comando do Windows: para poder se mover entre as pastas e ir para o local onde você irá trabalhar.

Caso você não tenha os conhecimentos desejáveis, nós providenciamos um passo a passo durante os primeiros exemplos. Basta acompanhar os textos e as figuras com cópias das telas.

1.3 Metodologia utilizada

Durante a escrita deste livro, procurei seguir a sequencia da maioria dos livros de introdução a programação e algoritmos. Contudo, verifiquei que existem pelo menos dois grupos de livros de informática: o primeiro grupo, o mais tradicional, procura apresentar os tópicos obedecendo uma estrutura sequencial, iniciando com as estruturas mais básicas até chegar nas estruturas mais complexas da linguagem. Pertencem a essa categoria livros como "Conceitos de computação usando C++" (Cay Horstman), "C++ como programar" (Deitel e Deitel), "Princípios e práticas de programação com C++" (Bjarne Stroustrup) , "Clipper 5.0 Básico" (José Antonio Ramalho) e a coleção "Clipper 5" (Geraldo A. da Rocha Vidal). Nesses livros não existem muitas quebras de sequencia, por exemplo: os códigos envolvendo variáveis são muito poucos, até que o tópico variável seja abordado completamente. Essa abordagem pode não agradar a algumas pessoas, da mesma forma que um filme de ação pode entediar o expectador se demorar demais a exibir as cenas que envolvem batidas de carro e perseguições. Por outro lado, essa abordagem é a ideal para o estudante que deseja iniciar a leitura de um código entendendo exatamente tudo o que está lá. A grande maioria dos elementos que aparecem nos exemplos já foram abordados em teoria. Por outro lado, existem os livros do segundo grupo, tais como : "Clipper 5.2" (Rick Spence), "A linguagem de programação C++" (Bjarne Stroustrup) e "C: A linguagem de programação" (Ritchie e Kerninghan). Esses livros enfatizam a prática da programação desde o início e são como filmes de ação do tipo "Matrix", que já iniciam prendendo a atenção de quem assiste. Essa abordagem agrada ao estudante que não quer se deter em muitos detalhes teóricos antes de iniciar a prática.

O presente livro pertence aos livros do primeiro grupo. Desde o início foi estimulada a prática da programação com vários códigos, desafios e exercícios, sem"queimar etapas". Algumas exceções podem ser encontradas, por exemplo, ainda no primeiro contato com a linguagem alguns conceitos avançados são vistos, mas de uma forma bem simples e sempre com um aviso alertando que esse tópico será visto mais adiante com detalhes. Os livros que pertencem aos do segundo grupo são ótimos, inclusive citei três nomes de peso que escreveram obras

desse grupo : Dennis Ritchie (criador da linguagem C), Bjarne Stroustrup (criador da linguagem C++) e Rick Spence (um dos desenvolvedores da linguagem Clipper). Porém procurei não fugir muito da sequencia dos cursos introdutórios de programação. Se por acaso você não tem experiência alguma com programação e deseja iniciar o aprendizado, essa livro será de grande ajuda para você, de modo que você não terá muitas surpresas com os códigos apresentados. Se você praticar os códigos apresentados, o aprendizado será um pouco lento, mas você não correrá o risco de ter surpresas no seu futuro profissional por usar um comando de uma forma errada. Caso você se desestimule durante o aprendizado isso não quer dizer que você não tem vocação para ser um programador, nem também quer dizer que eu não sei escrever livros de introdução a programação. Não se sinta desestimulado caso isso aconteça, parta para os livros que estimulam uma abordagem mais prática, sem fragmentos isolados de código ou tente outra vez com outra publicação do primeiro grupo.

Portanto, esse livro pretende ser um manual introdutório de programação, ele presume que você não tem conhecimento algum dos conceitos básicos, tais como variáveis, operadores, comandos, funções, loops, etc.

1.4 Material de Apoio

Esse livro baseia-se em uma simples ideia : programar é uma atividade que demanda muita prática, portanto, você será estimulado desde o início a digitar todos os exemplos contidos nele. É fácil achar que sabe apenas olhando para um trecho de código, mas o aprendizado real só ocorre durante o ciclo de desenvolvimento de um código. Digitar códigos de programas lhe ajudará a aprender o funcionamento de um programa e a fixar os conceitos aprendidos. Procurei também incluir no material de apoio o compilador Harbour que eu usei para escrever os exemplos desse livro, a linguagem C que acompanha o compilador Harbour e o editor de códigos xDevStudio, desenvolvido por Vailton Renato.

1.5 Plano da obra

O livro é organizado como se segue. O **capítulo 1** é esse capítulo, seu objetivo é passar uma ideia geral sobre as características do livro e dos métodos de ensino adotados. O **capítulo 2** ensina conceitos básicos de computação, compilação e geração de executáveis. O objetivo principal desse capítulo é lhe ensinar a instalar o ambiente de programação e criar alguns programas de teste para verificar se tudo foi instalado corretamente. O **capítulo 3** inicia os conceitos básicos de programação, o objetivo principal é permitir que você mesmo crie seus primeiros programas de computador. São abordados conceitos básicos de indentação, comentários, quebras de linha, padrão de codificação, strings, uso de aspas e estimula a busca de pequenos erros em códigos. O **capítulo 4** aborda as variáveis de memória, ensina alguns comandos básicos de como se receber os dados e trás alguns exemplos bem simples envolvendo operadores e funções básicas . O **capítulo 5** trás os tipos de dados da linguagem e seus respectivos operadores. Aborda também o valor NIL e os SETs da linguagem. Ele é importante, porém os exemplos ainda não são muitos desafiadores. O **capítulo 6** aborda um assunto central em

qualquer linguagem de programação: as estruturas de controle. Ele é um capítulo longo porque ele aborda juntamente com tais estruturas os conceitos de algoritmos e fluxogramas. Também lhe dá algumas técnicas de como transformar um problema do mundo real em um programa de computador. Ao finalizar esse capítulo você já poderá escrever pequenos programas realmente funcionais. O **capítulo 7** introduz o uso de funções. Ele divide-se em duas partes: a primeira trás uma pequena listagem das funções do Harbour e vários exemplos que devem ser praticados. A segunda parte desse capítulo lhe ensina a criar as suas próprias funções. O **capítulo 8** aborda as estruturas complexas da linguagem, como arrays e hashs. Se você acompanhou tudo direitinho até o capítulo 7 não haverá problemas aqui, pois a maioria dos tópicos que ele aborda já foram vistos. O **capítulo 9** aborda as classes de variáveis e o **capítulo 10** conclui o livro com alguns aspectos avançados da linguagem. O livro também possui uma lista de apêndices que podem ser consultados posteriormente, tais como modelos de fluxogramas, lista de SETs, resumo de programação estruturada, etc.

1.6 Porque Harbour ?

A linguagem Harbour é fruto da Internet e da cultura de software-livre. Ela resulta dos esforços de vários programadores, de empresas privadas, de organizações não lucrativas e de uma comunidade ativa e presente em várias partes de mundo. O Harbour surgiu com o intuito de preencher a lacuna deixada pela linguagem Clipper, que foi descontinuada e acabou deixando muitos programadores orfãos ao redor do mundo. Harbour é uma linguagem poderosa, fácil de aprender e eficiente, todavia ela peca pela falta de documentação e exemplos. A grande maioria das referências encontradas são do Clipper, esse sim, possui uma documentação detalhada e extensa. Como o Harbour foi feito com o intuito principal de beneficiar o programador de aplicações comerciais ele acabou não adentrando no meio acadêmico, o que acabou prejudicando a sua popularização. As universidades e a cultura acadêmica são fortes disseminadores de cultura, opinião e costumes. Elas já impulsionaram as linguagens C, C++, Pascal, Java, e agora, estão encantadas por Python. Por que então estudar Harbour ? Abaixo temos alguns motivos :

1. **Simplicidade e rapidez** : como já foi dito, Harbour é uma linguagem fácil de aprender, poderosa e eficiente. Esses três requisitos já constituem um bom argumento em favor do seu uso.

2. **Portabilidade** : Harbour é um compilador multi-plataforma. Com apenas um código você poderá desenvolver para o Windows, Linux e Mac. Existem também outras plataformas que o Harbour já funciona : Android, FreeBSD, OSX e até mesmo MS-DOS.

3. **Integração com a linguagem C** : internamente um programa escrito em Harbour é um programa escrito em C. Na verdade, para você ter um executável autônomo você necessita de um compilador C para poder gerar o resultado final. Isso confere a linguagem uma eficiência e uma rapidez aliada a uma clareza do código

escrito. Além disso é muito transparente o uso de rotinas C dentro de um programa Harbour. Caso você não queira um executável autônomo, e dispensar o compilador C, o Harbour pode ser interpretado[2] através do seu utilitário *hbrun*.

4. **Custo zero de aquisição** : Por ser um software-livre, o Harbour lhe fornece todo o poder de uma linguagem de primeira linha a custo zero.

5. **Constante evolução** : Existe um número crescente de desenvolvedores integrados em fóruns e comunidades virtuais que trazem melhorias regulares.

6. **Suporte** : Existem vários fóruns para o programador que deseja desenvolver em Harbour. A comunidade é receptiva e trata bem os novatos.

7. **Facilidades adicionais** : Existem muitos produtos (alguns pagos) que tornam a vida do desenvolvedor mais fácil, por exemplo : bibliotecas gráficas, ambientes RAD de desenvolvimento, RDDs para SQL e libs que facilitam o desenvolvimento para as novas plataformas (como Android e Mac OSX)

8. **Moderna** : A linguagem Harbour possui compromisso com o código legado mas ela também possui compromisso com os paradigmas da moderna programação, como Orientação a Objeto e programação Multithread.

9. **Multi-propósito** : Harbour possui muitas facilidades para o programador que deseja desenvolver aplicativos comerciais (frente de loja, ERPs, Contabilidade, Folha de pagamento, controle de ponto, etc.). Por outro lado, apesar de ser inicialmente voltada para preencher as demandas do mercado de aplicativos comerciais a linguagem Harbour possui muitas contribuições que permitem o desenvolvimento de produtos em outras áreas: como bibliotecas de computação gráficas, biblioteca de jogos, desenvolvimento web, funções de rede, programação concorrente, etc.

10. **Multi-banco** : Apesar de possuir o seu próprio banco de dados, o Harbour pode se comunicar com os principais bancos de dados relacionais do mercado (SQLite, MySQL, PostgreSQL, Oracle, Firebird, MSAccess, etc.). Além disso, todo o aprendizado obtido com o banco de dados do Harbour pode ser usado no bancos relacionais. Um objeto de banco de dados pode ser manipulado diretamente através de RDDs, tornando a linguagem mais clara. Você escolhe a melhor forma de conexão.

11. **Programação Windows** : Apesar de criticado por muitos, o Microsoft Windows repousa veladamente nos notebooks de muitos mestres e doutores em Ciência da Computação. A Microsoft está sentindo a forte concorrência dos smartphones e da web, mas ela ainda domina o mundo desktop. Com Harbour você tem total facilidade para o desenvolvimento de um software para windows. Além das libs gráficas para criação de aplicações o Harbour possui acesso a dlls, fontes ODBC,

[2]Nota técnica: o Harbour não pode ser considerada uma linguagem interpretada, pois um código intermediário é gerado para que ela possa ser executada. O código fonte não fica disponível como em outras linguagens.

ADO e outros componentes (OLE e ActiveX). Automatizar uma planilha em Excel, comunicar-se com o outros aplicativos que possuem suporte a windows é uma tarefa simples de ser realizada com o Harbour.

Existem pontos negativos ? Claro que sim. A falta de documentação atualizada e a baixa base instalada, se comparada com linguagens mais populares como o Java e o C#, podem ser fatores impeditivos para o desenvolvedor que deseja ingressar rapidamente no mercado de trabalho como empregado. Esses fatores devem ser levados em consideração por qualquer aspirante a programador. Sejamos claros : se você deseja aprender programação com o único objetivo de arranjar um emprego em pouco tempo então não estude Harbour, parta para outras linguagens mais populares como Java, PHP e C#. Harbour é indicado para os seguintes casos :

1. **O programador da linguagem Clipper** que deseja migrar para outra plataforma (o Clipper gera aplicativos somente para a plataforma MS-DOS) sem efetuar mudanças significativas no seu código fonte.

2. **Profissional que quer ser o dono do próprio negócio desenvolvendo aplicativos comerciais** : Sim, nós dissemos que você pode até desenvolver jogos e aplicativos gráficos com Harbour. Mas você vai encontrar pouco sobre esses assuntos nos fóruns da linguagem. O assunto predominante, depois de dúvidas básicas, são aqueles relacionados ao dia-a-dia de um sistema de informação comercial, como comunicação com impressoras fiscais e balanças eletrônicas, acesso a determinado banco de dados, particularidades de alguma interface gráfica, etc. O assunto pode, até mesmo, resvalar para normas fiscais e de tributação, pois o programador de aplicativos empresariais precisa conhecer um pouco sobre esses assuntos.

3. **Profissional que quer aprender rapidamente** e produzir logo o seu próprio software com rapidez e versatilidade.

4. **O usuário com poucos conhecimentos técnicos de programação** mas que já meche com softwares de produtividade (Excel, Calc, MS-Access, etc.) podem se utilizar do Harbour para adentrar no mundo da programação pela porta da frente em pouco tempo.

5. **O estudante** que quer ir além do que é ensinado nos cursos superiores. Com Harbour você poderá integrar seus códigos C e C++ (Biblioteca QT, wxWidgets, etc.) além de fuçar as “entranhas” de inúmeros projetos livres desenvolvidos em Harbour. Por trás do código simples do Harbour existem montanhas de código em C puro, ponteiros, estruturas de dados, integração com C++, etc.

6. **O aprendiz** ou o *hobbista* que quer aprender a programar computadores mas não quer adentrar em uma infinidade de detalhes técnicos pode ter no Harbour uma excelente ferramenta.

7. **O Hacker**[3] que quer desenvolver ferramentas de segurança (e outras ferramentas de análise) pode obter com o Harbour os subsídios necessários para as suas pesquisas.

Se você deseja aprender a programar e quer conhecer a linguagem Harbour, as próximas páginas irão lhe auxiliar nos primeiros passos.

[3]Hacker é um indivíduo que se dedica, com intensidade incomum, a conhecer e modificar os aspectos mais internos de dispositivos, programas e redes de computadores. Graças a esses conhecimentos, um hacker frequentemente consegue obter soluções e efeitos extraordinários, que extrapolam os limites do funcionamento "normal"dos sistemas como previstos pelos seus criadores. (Fonte : `https://pt.wikipedia.org/wiki/Hacker`. Acessado em 14-Ago-2016) O termo Hacker foi injustamente associado a crimes, invasões e ao esterótipo do jovem com reduzida atividade social. O termo usado para caracterizar os criminosos cibernéticos é *cracker*, suas atividades envolvem : desenvolvimento de vírus, quebra de códigos e quebra de criptografia. Outras categorias confundidas com hackers são : pichadores digitais, ciberterroristas, estelionatários, ex-funcionários raivosos, revanchistas e vândalos.

2 O PROCESSO DE CRIAÇÃO DE UM PROGRAMA DE COMPUTADOR

2.1 O que é um programa de computador ?

Para entender o processo de programação, você precisa entender um pouco sobre o que é um computador. O computador é uma máquina que armazena dados, interage com dispositivos e executa programas. Programas são instruções de sequencia e de decisão que o computador executa para realizar uma tarefa. Todos os computadores que você já utilizou, desde um relógio digital até um sofisticado smartphone são controlados por programas que executam essas operações extremamente primitivas. Parece impossível, mas todos os programas de computador que você já usou, todos mesmo, utilizam apenas poucas operações primitivas :

1. **entrada :** consiste em pegar dados do teclado, mouse, arquivo, sensor ou qualquer dispositivo de entrada;
2. **saída :** mostra dados na tela, imprime em duas ou três dimensões, envia dados para um arquivo ou outro dispositivo de saída.
3. **cálculo :** executa operações matemáticas simples.
4. **decisão :** avalia certas condições e executa uma sequencia de instruções baseado no resultado.
5. **repetir :** executa uma ação repetidamente.

Isso é praticamente tudo o que você precisa saber por enquanto. Nos próximos capítulos nós abordaremos cada um desse itens separadamente.

2.2 Linguagens de programação

Chamamos de programador o profissional responsável por dividir um problema da "vida real" em blocos de tarefas contendo as operações citadas na seção anterior. Para realizar esse intento ele utiliza uma linguagem de programação.

Antigamente, programar era uma tarefa muito difícil, porque ela se resumia a trabalhar diretamente com essas operações primitivas. Esse processo era demorado e enfadonho, pois o profissional tinha que entender como a máquina funcionava internamente para depois poder "entrar" com essas *instruções de máquina*. Essas instruções são codificadas como números, de forma que possam ser armazenados na memória. Como se não bastasse, elas diferiam de um modelo de computador para outro, e consistiam basicamente de números em sequencia, conforme o exemplo a seguir :

```
10011001 10011101 01101001 10011001 10011101 11111001
```

```
11111001 10011101 10001001 10011001 10011101 10011001
10011111 10011111 11111001 10011001 10011101 11111001
10011001 10011101 10000001 00011001 10011101 11100001
10011001 10011101 01111001 10011001 10011101 00111001
10011001 10011101 11111001 00011001 10011101 11111001
```

Ainda hoje os computadores, todos eles, funcionam através desses códigos. O que mudou foi a forma com a qual eles são gerados. Antes eles eram gerados diretamente por seres humanos e introduzidos diretamente na memória da máquina, através da alteração na posição de chaves ou "jumpers". Era uma tarefa tediosa e sujeita a erros.

Com o tempo, os cientistas desenvolveram um programa de computador cuja função era simplificar a criação desses códigos. Esse programa de computador, recebeu o nome de "montador" (*assembler*). Os montadores representaram um importante avanço sobre a programação em código de máquina puro, e a linguagem desse programa recebeu o nome de "Assembler". A sequencia abaixo representa um código em Assembler.

```
move int_rate, %eax
sub 100, %eax
jg  int_erro
```

A linguagem Assembler é classificada como uma linguagem de baixo nível. Essa classificação significa "o quão próximo da linguagem de máquina pura" uma linguagem de programação está. No caso do Assembler, a relação entre uma instrução em Assembler e uma instrução em código de máquina é praticamente de um para um. Os montadores também possuem "baixa portabilidade", ou seja, elas são dependentes do tipo de computador ao qual ele se destina. Se surgir um novo modelo de computador, o programa gerado em Assembler não irá funcionar.

Com o passar dos anos as linguagens de alto nível surgiram, tornando o processo de programar cada vez mais simples e acessível. O nome do programa usado para gerar o código de máquina através de uma linguagem não era mais montador, pois a relação entre o código gerado e o codigo de máquina não era mais de um para um, mas de um para "n". O nome do programa usado passou a ser "compilador", pois uma linha de código representa a compilação de várias linhas em linguagem de baixo nível. A linguagem Harbour é uma linguagem de alto nível, por exemplo a seguinte instrução :

```
a := 10
```

gera um código que equivale a várias linhas em uma linguagem de baixo nível. Outros exemplos de linguagens de alto nível : C, C++, Pascal, Visual Basic, Java, Clipper, COBOL, C#, etc. Existem também as linguagens de "médio nível" que, como o nome já sugere, é

um meio termo entre os dois tipos citados. Nessa categoria a Linguagem C é a única largamente utilizada[1].

O programador, sem saber, acaba desenvolvendo sempre em linguagem de máquina (que é a única linguagem que um computador entende), mas quando utiliza uma linguagem de alto nível, muitas operações são simplificadas e até ocultadas. É por isso que o termo "programa compilado em Assembler" não é tecnicamente correto, pois o nível com que um montador trabalha é diferente do nível com que um compilador trabalha.

2.3 Tipos de linguagens de programação

As linguagens de alto nível também se ramificaram quanto ao seu propósito. Por exemplo, a linguagem C é classificada como uma linguagem de propósito geral, ou seja, com ela nós podemos desenvolver aplicações para qualquer área. Existem programadores C que se especializaram na criação de sistemas operacionais, outros que desenvolvem software para controlar robôs industriais e outros que desenvolvem jogos de computador. Pode-se desenvolver praticamente de tudo com a linguagem C, mas todo esse poder tem um preço : ela é uma linguagem considerada por muitos como de difícil aprendizado. A linguagem Harbour deriva diretamente da linguagem C, mas ela não é assim tão genérica, e também não é de difícil aprendizado. O nicho da linguagem Harbour são aqueles aplicativos comerciais que compõem os sistemas de informação, desde pequenos sistemas para gerir uma pequena empresa até gigantescos ERPs podem ser construídos com Harbour. Você ainda tem o poder da linguagem C embutida nela, o que permite a criação de outros tipos de aplicativos, mas o foco dela é o ambiente organizacional.

Uma outra classificação usada é quanto a forma de execução do código. A grosso modo existem dois tipos de linguagens : a linguagem compilada e a linguagem interpretada[2]. Nós já vimos o que é uma linguagem compilada : o código digitado pelo programador é compilado em um código de máquina. As etapas que todo programador precisa passar para ter o seu código compilado pronto estão ilustrados na figura 2.1.

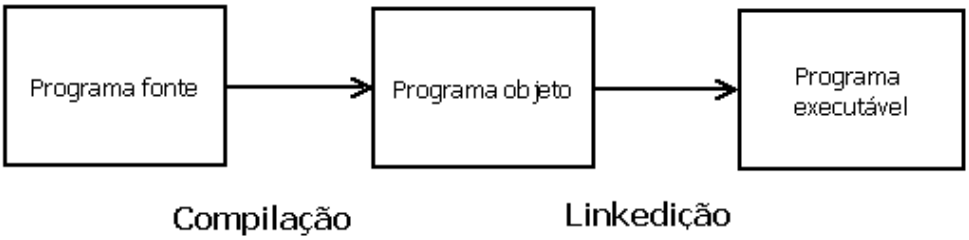
O que um compilador faz é basicamente o seguinte : lê todo o código que o programador criou e o transforma em um programa derivado, mas independente do código original. O código que o programador digitou é conhecido como "código fonte" e o código compilado é chamado de "código objeto". Depois que o código objeto é criado existe uma etapa final chamada de "ligação" (ou *linkedição*). O resultado final desse processo é um novo programa totalmente independente do compilador (esse arquivo é chamado de "executável"). As consequências práticas disso são :

1. O programa final é independente do compilador. Você não precisa instalar o compilador na máquina do cliente para o programa funcionar.

[1] Essa classificação de médio nível para a Linguagem C foi dada pelos criadores da linguagem. A Linguagem C, portanto, abrange duas categorias : alto nível e médio nível.

[2] Existem nuances entre essas duas classificações, mas ainda é cedo para você aprender sobre isso. Por hora, tudo o que você precisa saber são as diferenças básicas entre um compilador e um interpretador

Figura 2.1: O processo de compilação



2. O código fonte do programa não fica disponível para o usuário final. Isso confere ao programa uma segurança maior, pois ele não pode ser alterado por alguma pessoa não autorizada.

Um interpretador age de forma semelhante, mas não transforma o código fonte em um código independente. Em suma, ele sempre precisa da linguagem (interpretador) para poder funcionar. Os exemplos mais comuns são as linguagens PHP, ASP e Javascript. As consequências práticas disso são :

1. O programa final é dependente do interpretador (Você precisa ter o PHP instalado na sua máquina para poder executar programas em PHP, por exemplo)
2. O código fonte do programa fica disponível para o usuário final[3].

Via de regra, um programa compilado é bem mais rápido do que um programa interpretado, e mais seguro também. Ele é mais rápido porque ele é transformado em código de máquina apenas uma vez, durante o ciclo de compilação e linkedição. Já um programa interpretado sempre vai ser lido pela linguagem a qual ele pertence (por exemplo, toda vez que um programa em PHP é executado pelo usuário, ele é lido e checado linha por linha pelo interpretador PHP). Os programas compilados possuem a desvantagem de não serem facilmente portáveis [4] entre plataformas. É mais fácil escrever um programa PHP que funcione em servidores Windows e em Linux do que escrever um programa em C que funcione nessas duas plataformas citadas. Quando o programa é pequeno a dificuldade praticamente inexiste, mas quando o programa começa a ficar complexo o programador tem que se preocupar mais com a questão da portabilidade.

[3]Existem programas adicionais que podem obscurecer o código final, mas são etapas adicionais. Via de regra um programa interpretado tem o seu código fonte disponível, se não for protegido por essas ferramentas adicionais

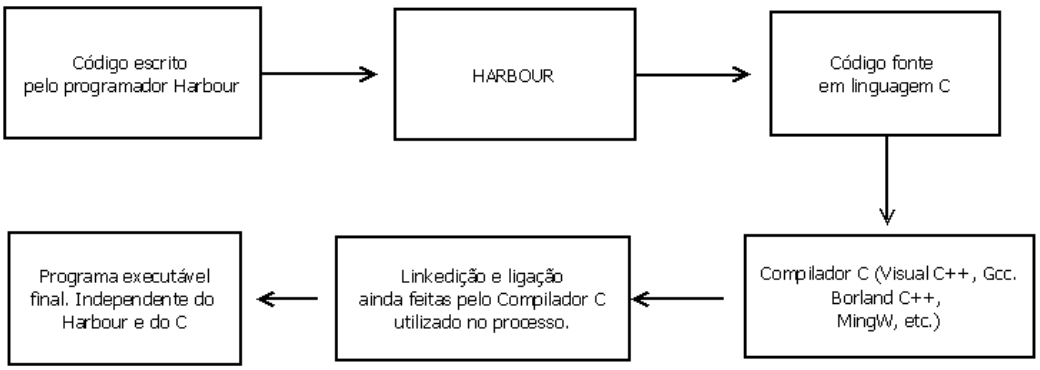
[4]Um programa portável pode funcionar sem problemas em diversos sistemas operacionais, como Windows, Linux, MAC, Android, etc.

2.3.1 O Harbour pertence a qual categoria ?

O lema da linguagem Harbour é : "escreva uma vez, compile em qualquer lugar", então o Harbour é um compilador certo ? Mais ou menos. Na verdade o Harbour não é classificado tecnicamente como um compilador puro, pois ele depende de um compilador externo para poder gerar o seu programa executável independente. Se você quer criar um programa executável 100% independente você vai precisar de um compilador C instalado na sua máquina. Isso porque o Harbour gera um outro código fonte que é lido posteriormente pelo compilador C para só depois gerar o código executável. Tecnicamente falando, todo programa executável gerado pelo Harbour é, na verdade, um executável feito em linguagem C. O programador Harbour, mesmo sem precisar aprender a linguagem C, gera o resultado final do seu trabalho através da linguagem C.

Todo esse processo de geração é simplificado para o programador, mas pode ser visualizado através da figura 2.2.

Figura 2.2: Processo de geração de um programa escrito em Harbour



Se você não tiver a linguagem C na sua máquina você ainda assim vai conseguir gerar um programa em Harbour que funcione, mas ele vai depender de um programa externo chamado *hbrun* para poder funcionar. É mais ou menos o mesmo processo que acontece com as linguagens interpretadas, a diferença é que o código que irá ser lido pelo *hbrun* não é o código fonte original do programador, mas um código objeto gerado pelo Harbour, dessa forma o código fonte de uma aplicação não precisa ser distribuída [5].

2.3.2 Instalando o Harbour no seu computador

O material que acompanha o livro pode ser baixado de `www.altersoft.net`. Ele vem em formato zip e o seu nome é "Curso_Harbour.zip". Descompacte esse conteúdo em uma

[5] Semelhante a forma com a qual um programa escrito em Java é gerado

pasta qualquer do seu sistema windows [6]. Como vamos seguir um roteiro de instalação, vamos descompactar essa pasta na raiz do seu sistema de arquivos windows, assim :

`C:\Curso_Harbour`.

Quando você descompactar o arquivo verifique se as seguintes pastas existem :

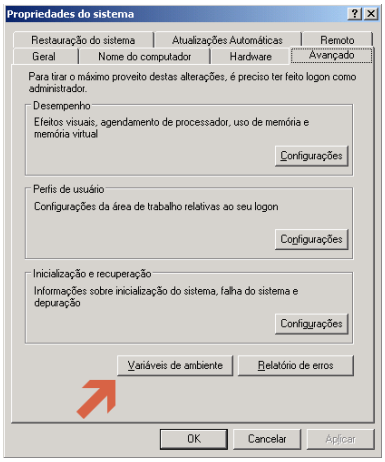
1. `C:\Curso_Harbour\harbour` : Pasta onde está o compilador harbour;
2. `C:\Curso_Harbour\xDevStudio` : Pasta onde está o editor xDevStudio;
3. `C:\Curso_Harbour\pratica` : Pasta onde você irá trabalhar digitando os códigos;

Para que o Harbour funcione corretamente tudo o que você tem a fazer é adicionar os PATHs de onde o Harbour está instalado e também de onde o GCC está instalado. Se você instalou no local padrão, que é `C:\Curso_Harbour` , então os PATHs são os seguintes :

- `C:\Curso_Harbour\harbour\bin`
- `C:\Curso_Harbour\harbour\comp\mingw\bin`

Para adicionar essas variáveis de ambiente faça conforme as figuras 2.3, 2.4 e 2.5

Figura 2.3: Adicionando as variáveis de ambiente



Quando for adicionar (figura 2.5) você deve ter o cuidado de não apagar as variáveis que estão lá. Vá até o final da linha, **digite um ponto e vírgula no final da linha** e digite o conteúdo baixo :

`C:\Curso_Harbour\Harbour\bin;C:\Curso_Harbour\Harbour\comp\mingw\bin`

Clique em Ok para salvar. Pronto, as variáveis estão configuradas.

[6] Se você usa o Linux, no apêndice XXX tem um roteiro de instalação no Linux

Figura 2.4: Adicionando as variáveis de ambiente

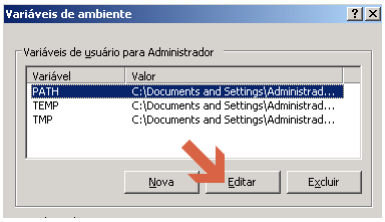
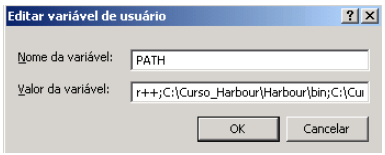


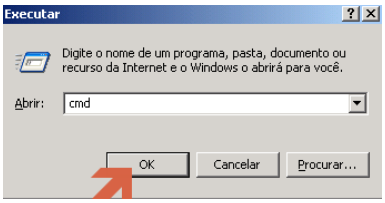
Figura 2.5: Adicionando as variáveis de ambiente



2.3.2.1 Testando se o Harbour foi instalado corretamente

Para testar faça o seguinte: abra o prompt do sistema operacional, conforme a figura 2.6.

Figura 2.6: Abrindo o prompt de comando do windows



No prompt digite “harbour” (sem as aspas). A figura logo abaixo é uma cópia das primeiras linhas que aparecem, mas a listagem é maior.

.:Resultado:.

```
Harbour 3.2.0dev (r1507030922)
Copyright (c) 1999-2015, http://harbour-project.org/

Syntax:  harbour <file[s][.prg]|@file> [options]

Options:  -a                automatic memvar declaration
          -b                debug info
          -build            display detailed version info
          -credits          display credits
          -d<id>[=<val>]    #define <id>
          -es[<level>]      set exit severity
          -fn[:[l|u]|-]     set filename casing (l=lower u=upper)
```

```
-fd[:[l|u]|-]     set directory casing (l=lower u=upper)
-fp[:<char>]      set path separator
-fs[-]            turn filename space trimming on or off
   (default)
```

Faça a mesma coisa com o compilador da linguagem C: Digite gcc e tecle enter.

.:Resultado:.

```
gcc: fatal error: no input files
compilation terminated.
```

Pronto, o compilador gcc também está instalado.

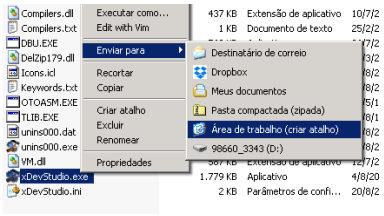
2.3.3 Instalando o xDevStudio

O xDevStudio é um editor desenvolvido por Vailton Renato (WebNet do Brasil) especialmente para os desenvolvedores Clipper, Harbour e outros da família xBase. Nós insistimos para que você use esse editor para digitar os códigos desse livro. Por vários motivos :

1. Esse foi o mesmo editor utilizado para a digitação dos códigos de exemplo.
2. Usando esse editor você não terá problemas com a exibição de acentos e cedilhas.
3. Esse editor foi especialmente configurado para suportar todos os comandos do Harbour. Por exemplo, quando você digitar alguma instrução de programação conhecida, essa instrução irá ser destacada com uma cor, e as vezes com um formato diferente.
4. O editor é totalmente em português, desenvolvido por um programador brasileiro e que não deve em nada aos similares estrangeiros.
5. O editor possui ótimos recursos de documentação da linguagem Clipper / Harbour. Acompanha os manuais do Clipper (que são todos válidos para o Harbour também), além de outras facilidades. Nessa parte alguns documentos podem estar em Inglês.

Portanto, o xDevStudio é o editor recomendado para que você use para digitar os códigos desse livro. Nós não chegaremos a utilizar todos os recursos do xDevStudio, por exemplo, a compilação através do editor não será usada, pois nós queremos que você aprenda a compilar um programa pelo prompt de comando do windows. Depois, quando você entender o que está ocorrendo você pode partir, em uma etapa posterior a leitura desse livro, para estudar o xDevStudio a fundo. É bom ressaltar que existem também outras opções free para o desenvolvimento de programas em Harbour, mas por enquanto, você deve usar o xDevStudio.

Figura 2.7: Criando um atalho na área de trabalho



Para facilitar o acesso ao editor faça conforme a figura 2.7

Clique com o botão direito do mouse sobre o icone do xDevStudio e crie um atalho na área de trabalho. Vamos agora seguir adiante no nosso aprendizado.

Dê agora um duplo click sobre o ícone do xDevStudio. Aguarde o carregamento do editor. Após o carregamento do editor, acesse o prompt do windows através do xDevStudio. Clique no menu “Ferramentas” e selecione a opção “Prompt de Comando” conforme a figura 2.8. Aguarde um pouco e a janela do Prompt de comando irá aparecer.

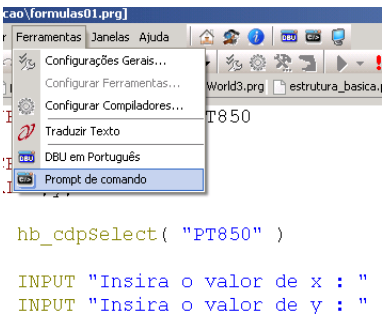
Observe que o prompt de comando abre na mesma pasta onde está o xDevStudio. Tudo o que você tem a fazer é navegar, através de comandos do prompt até a pasta de exemplos para poder compilar alguns exemplos de teste. Faça assim, dentro do prompt de comando (após cada linha tecle enter) :

.:Resultado:.

```
C:\Curso_Harbour\xDev> cd ..
C:\Curso_Harbour> cd pratica
C:\Curso_Harbour\pratica>
```

Pronto, você agora está na pasta pratica. Nessa pasta nós iremos testar o compilador. Utilize também essa pasta para digitar os seus códigos.

Figura 2.8: Abrindo o prompt de comando do windows pelo xDevStudio



2.3.4 O processo de compilação

Bem, se tudo correu bem até agora você deve ter :

1. O compilador Harbour instalado
2. O compilador C (gcc) instalado
3. O xDevStudio com um atalho na área de trabalho
4. O prompt de comando aberto e na pasta pratica

Agora vamos entender o processo de compilação de um programa em Harbour.

Para compilar um programa em Harbour existe um utilitário chamado *hbmk2* que faz todo o processo descrito no capítulo anterior. Esse programa não faz a compilação, mas gerencia o Harbour e todos os programas envolvidos durante o processo. Digite no prompt de comando *hbmk2* e tecle enter. Algo semelhante com a figura abaixo deve surgir (não mostramos tudo na figura abaixo, apenas as linhas iniciais).

.:Resultado:.

```
Harbour Make (hbmk2) 3.2.0dev (r2015-07-03 09:22)
Copyright (c) 1999-2013, Viktor Szakáts
http://harbour-project.org/
Translation (pt-BR): Vailton Renato <vailtom@gmail.com>
```

Dentro da pasta “pratica” existe um arquivo chamado exemplo.prg, agora iremos compilar esse arquivo.

Digite hbmk2 exemplo e tecle enter [7]

.:Resultado:.

```
hbmk2: Processando script local: hbmk.hbm
hbmk2: Harbour: Compilando módulos...
Harbour 3.2.0dev (r1507030922)
Copyright (c) 1999-2015, http://harbour-project.org/
Compiling 'exemplo.prg'...
Lines 15, Functions/Procedures 1
Generating C source output to '.hbmk\win\mingw\exemplo.c'... Done.
hbmk2: Compilando...
hbmk2: Linkando... exemplo.exe
```

O processo de compilação é simples e rápido, mas durante esse processo, muitas operações aconteceram. Vamos analisar a saída do comando hbmk2.

- **hbmk2: Processando script local: hbmk.hbm** : O sistema de compilação do Harbour possui inúmeras opções de linha de comando. Não vamos nos aprofundar nessas opções por enquanto. O que queremos salientar é que essas opções

[7] Com o passar dos exemplos nós omitiremos detalhes básicos, como teclar enter após a digitação no prompt de comando.

podem ficar arquivadas em um arquivo chamado *hbmk.hbm*. Nós já providenciamos esse arquivo com algumas das opções mais usadas, por isso o compilador exibiu essa mensagem.

- **Compiling 'exemplo.prg'...** : A fase de compilação do Harbour.
- **Lines 15, Functions/Procedures 1** : Ele analisou o arquivo e encontrou 15 linhas e uma Procedure ou Função (mais na frente veremos o que é isso).
- **Generating C source output to ...** : Ele informa onde gerou o arquivo para o compilador C trabalhar. Não devemos nos preocupar com esse arquivo. O Harbour mesmo o coloca em um lugar isolado, dentro de uma pasta que ele mesmo criou.
- **hbmk2: Compilando...** : A fase onde o compilador C entra em ação.
- **hbmk2: Linkando... exemplo.exe** : A fase final, de linkagem e consequente geração do executável.

2.3.5 Conclusão

Pronto, concluímos um importante processo no entendimento de qualquer linguagem de programação. No futuro, se você for estudar outra linguagem que use um compilador, os processos serão bem semelhantes. O próximo passo é a geração do seu primeiro programa em Harbour.

3 MEU PRIMEIRO PROGRAMA EM HARBOUR

3.1 De que é feito um programa de computador ?

Todo programa de computador constitui-se de uma sequência lógica de instruções que realizam tarefas específicas. Essas sequências ficam agrupadas dentro de conjuntos ou blocos de instruções chamados de "rotina". Nós, seres humanos, costumamos usar a expressão "hábito" para expressar uma rotina qualquer de nossas vidas, por exemplo : escovar os dentes após acordar pela manhã, vestir a roupa para ir trabalhar, dirigir um carro, etc.

Dentro de cada rotina nossa existem ações que executamos em uma sequência lógica. Da mesma forma, um programa de computador é composto por um conjunto de rotinas, cada uma delas engloba um bloco de instruções que são executados sequencialmente. Se nós fossemos um computador, a rotina "trocar uma lâmpada" poderia ser descrita assim :

```
ROTINA Trocar uma lâmpada

   1. Pegar uma escada.
   2. Posicionar a escada embaixo da lâmpada.
   3. Buscar uma lâmpada nova
   4. Subir na escada
   5. Retirar a lâmpada velha
   6. Colocar a lâmpada nova

FINAL DA ROTINA
```

3.2 O mínimo necessário

A linguagem Harbour possui dois tipos de rotinas : as *procedures* e as *funções*. Mais na frente, dedicaremos um capítulo inteiro ao estudo dessas rotinas, mas por enquanto, tudo o que você deve saber é que esses blocos de instruções agrupam todas os comandos de um programa de computador. Uma analogia interessante é comparar o desenvolvimento de um programa a leitura de um romance: a grande maioria dos romances está organizada em capítulos. Quando nós vamos iniciar a leitura começamos pelo capítulo 1 e seguimos em sequencia até o final do livro. Em um programa de computador cada capítulo corresponde a uma rotina, porém a ordem com que essas rotinas são executadas (a ordem de leitura) é determinada por uma rotina especial chamada de "rotina principal" [1].

De acordo com (DAMAS, 2013, p. 9) um programa é uma sequência lógica organizada de tal forma que permita resolver um determinado problema. Dessa forma terá que existir

[1] Em inglês a tradução da palavra "principal" é "main".

um critério ou regra que permita definir onde o programa irá começar. Esse "critério", o qual Luis Damas se refere, é a existência da rotina principal. Mas como o computador sabe qual é a rotina principal ?

No caso do Harbour, existe uma rotina (função ou procedure) onde são colocadas todas as instruções que devem ser executadas inicialmente. Essa rotina [2] chama-se *Main*, e todo bloco a executar fica entre o seu cabeçalho (PROCEDURE Main) e o comando Return.

Listing 3.1: O primeiro programa

```
1 PROCEDURE Main
2
3
4 RETURN
```

O programa escrito em 3.1 é completamente funcional do ponto de vista sintático, porém ele não executa nada. Com isso queremos dizer que o mínimo necessário para se ter um programa sintaticamente correto é a procedure Main.

3.3 O primeiro programa

Vamos agora criar o nosso primeiro programa em Harbour. O exemplo a seguir (listagem 3.2) exibe a frase "Hello World" na tela. O símbolo "?" é um comando que avalia o conteúdo da expressão especificada e o exibe.

Listing 3.2: Hello World

```
1 PROCEDURE Main
2
3     ? "Hello World"
4
5 RETURN
```

Digite o programa usando o xDevStudio e compile o programa usando o *hbmk2*.

Abra o xDevStudio, clique no menu Arquivo, e selecione a opção Novo e em seguida Arquivo PRG. Todo programa escrito em Harbour possui a extensão PRG. Inicie a digitação do programa da listagem 3.2, conforme a figura 3.1.

Quando concluir a digitação salve o arquivo. Como é a primeira vez que você salva esse arquivo você deve informar o nome e o local onde ele ficará. Salve-o na pasta pratica, conforme a figura 3.2 e 3.3. Quando for salvar dê-lhe o nome "hello.prg".

Agora abra o Prompt de comando e vá para a pasta pratica. Digite o comando para gerar o programa. Note que não precisa digitar a extensão do arquivo.

[2] Note que nos exemplos nós usamos a palavra reservada PROCEDURE para iniciar a nossa rotina. Nos capítulos seguintes nós veremos a diferença entre uma procedure e uma função.

Figura 3.1: Abrindo o prompt de comando do windows pelo xDevStudio

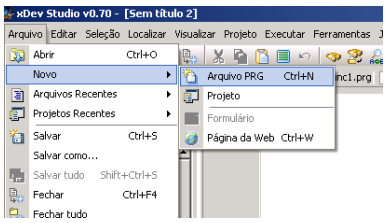
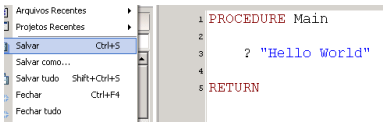


Figura 3.2: Salvando o arquivo



.:Resultado:.

```
C:\Curso_Harbour\pratica> hbmk2 hello
```

Após o processo de compilação, que não deve demorar, você deve executar o programa digitando o seu nome e teclando enter. Conforme a figura abaixo :

.:Resultado:.

```
C:\Curso_Harbour\pratica> hello
```

O programa deve exibir a saída :

.:Resultado:.

```
Hello World
```

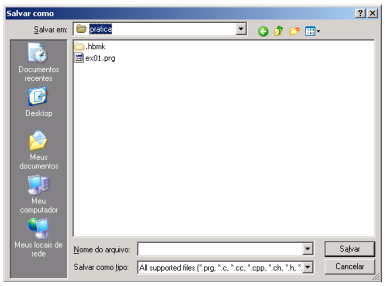
Pronto, você gerou o seu primeiro programa em Harbour. Use essa seção como tira-dúvidas pois nós vamos omitir todos esses passos e telas quando formos compilar os outros exemplos. Nos demais exemplos nós apenas mostraremos a listagem para você digitar e uma cópia simples da tela (sem os detalhes do Prompt de Comando) com a respectiva saída.

Apesar de pequeno, esse programa nos diz algumas coisas.

1. Na linha 1 temos o início do programa que é indicado pelo simbolo **PROCEDURE Main**.

2. Na linha 3 temos o símbolo "?" que é o responsável pela exibição de dados na tela do computador.

Figura 3.3: Salvando o arquivo (Parte II)



3. Ainda na linha 3 nós temos a frase “Hello World”. Que é o conteúdo que será exibido na tela.

Faça algumas experiências: tente recompilar o programa da listagem 3.2 mas com algumas pequenas alterações. Os criadores de uma das mais utilizadas linguagens de programação (a linguagem C) afirmam que o caminho para se aprender uma linguagem qualquer é treinar escrevendo códigos. Eles acrescentam que devemos experimentar deixar de fora algumas partes do programa e ver o que acontece (KERNIGHAN; RITCHIE, 1986, p. 20). Vamos então seguir essa dica e realizar algumas alterações. Na lista abaixo nós temos cinco dicas de alterações que podemos realizar, apenas tome o cuidado de fazer e testar uma alteração por vez. Não realize duas ou mais alterações para verificar o que mudou pois você poderá não saber o que causou o comportamento. Vamos tentar, então de um por um:

1. Na linha 1 troque a palavra Main por mAin.
2. Na linha 3 troque a frase “Hello World” por outra frase qualquer.
3. Na linha 3 escreva novamente “Hello World”, mas não coloque o último parêntese.
4. Na linha 5 troque RETURN por RETORNO
5. Na linha 5 troque RETURN por RETU

Como é a primeira vez, vamos a seguir comentar os resultados desse teste e acrescentar algumas explicações adicionais.

1. **Teste 1:** Nada aconteceu. Isso porque a linguagem Harbour não faz distinção entre maiúsculas e minúsculas. Linguagens assim são chamadas de **Case Insensitive**, isto é, a linguagem **não faz** diferenciação entre letras maiúsculas e minúsculas, sendo portanto a mesma coisa escrever main, Main, mAin ou MAIN.
2. **Teste 2:** O conteúdo que está entre parênteses é livre para você escrever o que quiser, e a linguagem Harbour irá exibir **exatamente o que está lá**. Se você escreveu tudo com letras maiúsculas, então o computador irá exibir tudo em maiúsculo, e assim por diante. É a única exceção da observação anterior.

3. **Teste 3:** Se você abriu um parenteses mas não fechou (não importa se foi o primeiro ou o último) ou se você esqueceu os parênteses, então o programa não irá ser gerado. Um erro será reportado.

4. **Teste 4:** Se você trocou RETURN por RETORNO o programa irá exibir um erro e não será gerado. Isso significa que os símbolos que **não estão** entre aspas exigem uma digitação correta deles.

5. **Teste 5:** Se você trocou RETURN por RETU o programa irá funcionar perfeitamente. Isso acontece porque **você não precisa escrever o nome do comando completamente**. Bastam os quatro primeiros dígitos. **Mas não faça isso nos seus programas**. Habitue-se a escrever os nomes completos, pois isso facilitará o seu aprendizado, além de tornar os seus programas mais claros para quem os lê[3]

Note que, se você fez o teste, algumas alterações ficaram sem explicação. Por exemplo, no item cinco, a troca de RETURN por RETU não alterou em nada a geração do programa, mas você provavelmente não conseguiu explicar a causa. Somente com a leitura adicional você ficou sabendo que a causa é uma característica da linguagem Harbour que emula as antigas linguagens Clipper e dBase. O que nós queremos dizer com isso é que: **muito provavelmente você não irá conseguir explicar a causa de todas as suas observações sozinho**. Alguém terá que lhe contar. Mas isso não deve lhe desanimar, pois enquanto estamos descobrindo uma linguagem nova, muitas perguntas ficarão sem resposta até que você avance alguns capítulos e descubra a causa.

Dica 1

Habitue-se a digitar o nome completo das funções e dos comandos Harbour.

Dica 2

Note que o código da listagem 3.2 possui a parte central "recuada" em relação ao início (*PROCEDURE Main*) e o fim do bloco (*RETURN*). Esse recuo chama-se "indentação"[a] e cada marca de tabulação (obtida com a tecla Tab) representa um nível de indentação. Essa pratica não é nenhuma regra obrigatória, mas ajuda a manter o seu código legível. Tome cuidado com o editor de texto que você utiliza para programar, pois os espaços entre as tabulações podem variar de editor para editor. Fique atento ao tamanho das tabulações no seu editor de texto, as tabulações são úteis para facilitar a marcação das indentações. Geralmente os editores possuem uma opção [b] que permite alterar o tamanho da tabulação. Alguns editores possuem ainda uma opção extra que converte as tabulações automaticamente em espaços, e nós consideramos uma boa ideia habilitar essa opção, caso ela exista. Nos exemplos desse livro as tabulações possuem o tamanho igual a três

[3]Essa característica estranha da linguagem Harbour nasceu do seu compromisso de ser compatível com a linguagem Clipper. O Clipper, assim como o dBase, que nasceu na década de 1980, podia reconhecer um comando apenas pelos quatro primeiros caracteres. Essa característica tornava o texto dos programas (o código fonte) menor e economizava espaço em disco. Hoje em dia não precisamos mais dessa economia porque os dispositivos de armazenamento de hoje possuem uma capacidade muito superior aos antigos disquetes.

espaços.

[a]Essa palavra é um neologismo que tem a sua origem na palavra inglesa “indentation”. Maiores detalhes em http://duvidas.dicio.com.br/identacao-ou-indentacao/ e em https://en.wikipedia.org/wiki/Indentation_(typesetting) (Acessado em 20-Ago-2016)

[b]Na maioria dos editores essa opção fica no menu *Preferences* , *Preferências* ou *Opções gerais*.

Dica 3

Se você for do tipo que testa tudo (o ideal para quem quer realmente aprender programação), você deve ter percebido que eu menti quando disse que o mínimo necessário para se ter um programa sintaticamente completo é a PROCEDURE Main (Listagem 3.1). Na verdade, um programa é gerado mesmo sem a procedure Main. Mesmo assim você deve se habituar a criar a procedure Main, pois **isso irá garantir que o programa irá iniciar por esse ponto (o Harbour procura por uma procedure com esse nome, e quando acha inicia o programa por ela.)**. Caso você não crie essa procedure, você correrá o risco de não saber por onde o seu programa começou caso ele fique com muitos arquivos. Portanto : **crie sempre uma procedure Main nos seus programas**, mesmo que eles tenham apenas poucas linhas. Isso irá lhe ajudar a criar um bom hábito de programação e lhe poupará dores de cabeça futuras. Programas sem a rotina principal não constituem bons exemplos para o dia a dia. Como nós queremos começar corretamente, vamos sempre criar essa procedure.

A seguir (listagem 3.3) temos uma pequena variação do comando “?”, trata-se do comando “??”. A diferença é que esse comando não emitirá um *line feed* (quebra de linha) antes de exibir os dados, fazendo com que eles sejam exibidos na linha atual. O efeito é o mesmo obtido com o programa da listagem 3.2.

Listing 3.3: O comando ??

```
1 PROCEDURE Main()
2
3    ?? "Hello "
4    ?? "World"
5
6 RETURN
```

.:Resultado:.

```
Hello World
```

Usaremos esses dois comandos para exibir os dados durante esse livro pois é a forma mais simples de se visualizar um valor qualquer. Esses dois comandos admitem múltiplos valores separados por uma vírgula, conforme a listagem 3.4.

Listing 3.4: O comando ? e ??

```
1
```

```
2  /*
3  Comandos ? e  ??
4  */
5  PROCEDURE Main
6
7     ? "João chegou cedo hoje ", "mas Tainá chegou atrasada."
8     ? "O valor da compra foi de R$ ", 4, " reais"
9
10
11 RETURN
```

.:Resultado:.

```
Jo|o chegou cedo hoje  mas Tain| chegou atrasada.
O valor da compra foi de R$           4  reais
```

3.3.1 Uma pequena pausa para a acentuação correta

Se você digitou, compilou e executou o programa da listagem 3.4 e não utilizou o editor que recomendamos no capítulo anterior (o xDev), provavelmente você teve problemas com a exibição de palavras acentuadas. Você deve ter notado que a acentuação não apareceu corretamente e no lugar da palavra surgiu um símbolo estranho. Isso aconteceu devido a codificação incompatível entre o editor que você utilizou e a linguagem Harbour. Esse problema pode ser resolvido, mas nós não queremos abordar agora assuntos que só serão plenamente compreendidos mais adiante. Então, sem prejuízo para o aprendizado, vamos usar o editor indicado. Caso você queira explicações adicionais sobre a acentuação, você pode consultar o apêndice **??**.

Outra alteração que você deverá fazer nos seus programas é inserir as seguintes instruções no seu código:

1. *REQUEST HB_CODEPAGE_PT850*
2. *hb_cdpSelect("PT850")*

Listing 3.5: Corrigindo os acentos

```
1
2  /*
3  Comandos ? e  ??
4  */
5  REQUEST HB_CODEPAGE_PT850
6  PROCEDURE Main
7
8     hb_cdpSelect( "PT850" )
```

```
9     ? "João chegou cedo hoje ", "mas Tainá chegou atrasada."
10    ? "O valor da compra foi de R$ ", 4, " reais"
11
12
13  RETURN
```

.:Resultado:.

```
João chegou cedo hoje  mas Tainá chegou atrasada.
O valor da compra foi de R$           4  reais
```

A instrução *REQUEST HB_CODEPAGE_PT850* não é lido pelo programa durante a execução dele. Essa instrução é inserida apenas uma vez "em tempo de compilação" [4]. Por esse motivo essa instrução está antes da *PROCEDURE Main*. O programa continua iniciando a partir de *Main* conforme havíamos dito no capítulo anterior. Não vamos explicar agora como esse processo funciona, mas você pode consultar o apêndice XXX para entender o que é uma codificação de arquivo. **O que importa agora é que nós usaremos essas duas instruções nas nossas listagens futuras para que os acentos apareçam corretamente**.

Dica 4

Habitue-se a criar uma lista de trechos de códigos prontos para poupar tempo na digitação de trechos repetitivos. Esses trechos de códigos são conhecidos como *Snippets* [a]. Um *snippet* é simplesmente um trecho de código que nós salvamos em um arquivo e para para economizar tempo e trabalho na digitação. Um exemplo de um *snippet* :

```
1  REQUEST HB_CODEPAGE_PT850
2
3  PROCEDURE Main
4
5       hb_cdpSelect( "PT850" )
6       /* Inicie o seu programa a partir dessa linha */
7
8
9  RETURN
```

Portanto: habitue-se a criar sempre seus próprios trechos de código com passagens repetitivas. Você pode usar o trecho acima para facilitar a digitação.

[a] A palavra *snippet* surgiu como um recurso que a Microsoft disponibilizou nos seus editores de código. Porém, a ideia não é da Microsoft. Ela apenas automatizou um bom hábito que todo programador deve ter.

[4] A expressão "em tempo de compilação" quer dizer que foi durante a fase da geração do código objeto. Maiores detalhes sobre isso no capítulo anterior.

3.4 Uma pequena introdução as Funções

Na seção anterior nós também usamos o simbolo *hb_cdpSelect("PT850")* para exibir a acentuação corretamente. O nome desse símbolo é *função*. Vamos aprender agora um pouco sobre esse conceito novo que só será visto detalhadamente nos capítulos posteriores. Por enquanto vamos ficar com o seguinte conceito pouco formal : **uma função é um símbolo que possui um valor pré-determinado**.

A seguir temos um exemplo de uma função. Note que a presença de parênteses após o seu nome é obrigatória para que o Harbour saiba que se trata de uma função.

```
1 REQUEST HB_CODEPAGE_PT850
2
3 PROCEDURE Main
4
5     hb_cdpSelect( "PT850" )
6     ? ‘‘Olá pessoal!’’
7     ? TIME()  //  Imprime a hora
8
9  RETURN
```

Observe que na linha 6 do exemplo anterior, o comando “?” imprime “Olá pessoal”, mas na linha 7 o comando “?” não imprime a palavra “TIME()”. Quando o programa chega na linha 7 ele identifica que o símbolo é uma função conhecida e substitui o seu nome pelo valor que ela “retorna”, que é a hora corrente do sistema operacional.

.:Resultado:.

```
Olá pessoal!
12:29:30
```

Se por acaso a função TIME() tivesse sido digitada entre aspas, então o seu nome é que seria exibido na tela, e não o seu valor, conforme o exemplo a seguir :

```
1 REQUEST HB_CODEPAGE_PT850
2
3 PROCEDURE Main
4
5     hb_cdpSelect( "PT850" )
6     ? ‘‘TIME()’’
7     ? TIME()  //  Imprime a hora
8
9  RETURN
```

.:Resultado:.

```
TIME()
12:29:30
```

Se você quiser usar a função TIME() para mostrar a hora do sistema dentro de uma frase, como por exemplo “Agora são 10:37:45” você **não pode** fazer como os exemplos abaixo :

```
1 REQUEST HB_CODEPAGE_PT850
2
3 PROCEDURE Main
4
5     hb_cdpSelect( "PT850" )
6     ? ‘‘TIME()’’  // Mostra o nome TIME()
7     ? ‘‘A hora atual é ‘‘ TIME() // Gera um erro
8
9  RETURN
```

Para exibir a mensagem você deve fazer assim :

```
1 REQUEST HB_CODEPAGE_PT850
2
3 PROCEDURE Main
4
5     hb_cdpSelect( "PT850" )
6     ? ‘‘Agora são ‘‘, TIME()  // Use a vírgula do comando ‘‘?’’
7
8  RETURN
```

.:Resultado:.

```
Agora são  10:34:45
```

Esse comportamento vale para todas as funções. No próximo capítulo estudaremos essa característica detalhadamente e veremos também que existem formas melhores e mais indicadas de se fazer essa junção. Por enquanto, quando você for exibir uma função junto com uma frase, separe-as com uma vírgula.

A listagem 3.6 ilustra o uso da função SECONDS(), a “função” dela é “retornar” quantos segundos transcorreram desde a meia-noite.

Listing 3.6: Funções simples

```
1
2  /*
3  Função
4  */
5  REQUEST HB_CODEPAGE_PT850
6
7  PROCEDURE Main
8
9     hb_cdpSelect( "PT850" )
10    ? "Desde a meia-note até agora transcorreram ", SECONDS(), "
          segundos."
11
12 RETURN
```

.:Resultado:.

```
Desde a meia-note até agora transcorreram       62773.81  segundos.
```

3.4.1 Os delimitadores de strings

Até agora nós usamos frases entre aspas para serem exibidas com o comando "?", mas ainda não definimos essas tais frases. Pois bem, um conjunto de caracteres delimitados por aspas recebem o nome de *string*. Sempre que quisermos tratar esse conjunto de caracteres devemos usar aspas simples ou aspas duplas [5] , apenas procure não misturar os dois tipos em uma mesma *string* (evite abrir com um tipo de aspas e fechar com outro tipo, isso impedirá o seu programa de ser gerado).

As vezes é necessário imprimir uma aspa dentro de um texto, como por exemplo : "Ivo viu a 'uva"'. Nesse caso, devemos ter cuidado quando formos usar aspas dentro de strings. O seguinte código da listagem 3.7 está errado.

Listing 3.7: Erro com aspas

```
1  REQUEST HB_CODEPAGE_PT850
2
3  PROCEDURE Main()
4
5      hb_cdpSelect( "PT850" )
6      ? "A praia estava "legal""  // Errado
7      ? 'Demos um nó em pingo d'água' // Errado
8
9  RETURN
```

[5] O Harbour também utiliza colchetes para delimitar *strings*, mas esse recurso é bem menos usado. Prefira delimitar *string* com aspas (simples ou duplas).

Já o código da listagem 3.8 está correto :

Listing 3.8: Uso correto de delimitadores de *string*

```
/*
Delimitando strings
*/
REQUEST HB_CODEPAGE_PT850

PROCEDURE Main

    hb_cdpSelect( "PT850" )

    ? "A praia estava 'legal'."
    ? 'A praia estava "legal".'
    ? [Colchetes podem ser usados delimitador de strings!]
    ? [Você deram um "nó em pingo d'água"]
    ? "Mas mesmo assim evite o uso de colchetes."

RETURN
```

.:Resultado:.

```
A praia estava 'legal'.
A praia estava "legal".
Colchetes podem ser usados delimitador de strings!
Você deram um "nó em pingo d'água"
Mas mesmo assim evite o uso de colchetes.
```

3.5 Comentários

Os comentários são notas ou observações que você coloca dentro do seu programa mas que servem apenas para documentar o seu uso trabalho, ou seja, o programa irá ignorar essas linhas de comentários. Eles não são interpretados pelo compilador, sendo ignorados completamente.

O Harbour possui dois tipos de comentários : os de uma linha e os de múltiplas linhas.

Os comentários de uma linha são :

1. "NOTE" : Só pode ser usado antes da linha iniciar. (Formato antigo derivado do Dbase II, evite esse formato.).
2. && : Pode ser usado antes da linha iniciar ou após ela finalizar. (Formato antigo derivado do Dbase II, evite esse formato.).

3. “*” : Só pode ser usado antes da linha iniciar. (Formato antigo derivado do Dbase II. Alguns programadores ainda usam esse comentário, principalmente para documentar o início de uma rotina.).

4. “//”: Da mesma forma, pode ser usado antes da linha iniciar ou após ela finalizar (Inspirados na Linguagem C++, é o formato mais usado para comentários de uma linha).

Os comentários de múltiplas linhas começam com /* e termina com */, eles foram inspirados na Linguagem C. Esse formato é largamente utilizado.

Listing 3.9: Uso correto de comentários

```
1  REQUEST HB_CODEPAGE_PT850
2
3  PROCEDURE Main
4  LOCAL nVal // Útil para explicar a linha corrente.
5  // LOCAL cBox  <=== Essa linha TODA será ignorada.
6  * Essa linha também será ignorada.
7  && E essa também...
8
9     hb_cdpSelect( "PT850" )
10
11    ? "Olá, pessoal"
12    ? "O valor é " ,  1200,00  // Útil para comentar após
13                                     // o final da linha.
14
15 /*
16
17     Note que esse comentário
18     possui várias linhas.
19
20 */
21
22 RETURN
```

No dia-a-dia, usamos apenas os comentários “//” e os comentários de múltiplas linhas (/* */) mas, caso você tenha que alterar códigos antigos, os comentários “&&” e “*” são encontrados com bastante frequência.

O Harbour não permite a existência de comentários dentro de comentários. Por exemplo : /* Inicio /* Inicio2 */ Fim */.

Listing 3.10: Uso INCORRETO de comentários

```
1  PROCEDURE Main
2
3      /*
4              ERRO !
```

```
5                   Os comentários não podem ser aninhados, ou
6                   seja, não posso colocar um dentro do outro.
7
8                   /*  Aqui está o erro  */
9
10          */
11
12      ? "Olá pessoal, esse programa não irá compilar."
13
14  RETURN
```

Os comentários de múltiplas linhas também podem ser usados dentro de um trecho de código sem problema algum. Por exemplo :

```
x := a + b + c
```

é equivalente a :

```
x := a /* valor inicial */ + b /* valor bruto */ + c // Calculo de parcelas
```

Dica 5

Os comentários devem ajudar o leitor a compreender o problema abordado. Eles não devem repetir o que o código já informa claramente e também não devem contradizer o código. Eles devem ajudar o leitor a entender o programa. Esse suposto "leitor" pode até ser você mesmo daqui a um ano sem "mecher" no código. Seguem algumas dicas rápidas retiradas de (KERNIGHAN; PIKE, 2000, p. 25-30) sobre como comentar eficazmente o seu código :

1. Não reporte informações evidentes no código.
2. Sempre comente as funções, os dados globais e as classes. No caso de funções e procedimentos um comentário simples na abertura do bloco de código já é suficiente na maioria dos casos.
3. Quando o código for realmente difícil, como um algoritmo complicado ou uma norma fiscal obscura, procure indicar no código uma fonte externa para auxiliar o leitor (um livro com a respectiva página ou um site). Sempre cite referências que não corram o risco de "sumir" de uma hora para outra (evite blogs ou redes sociais, se não tiver alternativa salve esse conteúdo e tenha cópias de segurança desses arquivos).
4. Escolha nomes auto-explicativos para as suas variáveis, evite comentar variáveis com nomes esquisitos. Veremos mais a esse respeito adiante.

5. Não contradiga o código. Quando alterar o seu trabalho atualize também os seus comentários ou apague-os.

3.6 Quebra de linha

Linguagens como C, C++, Java, PHP, Pascal e Perl necessitam de um ponto e vírgula para informar que a linha acabou. Harbour não necessita desse ponto e vírgula para finalizar a linha, assim como Python e Basic. No caso específico da Linguagem Harbour, o ponto e vírgula é necessário para informar que a linha não acabou e deve ser continuada na linha seguinte.

Listing 3.11: A linha não acabou

```
1  REQUEST HB_CODEPAGE_PT850
2
3  PROCEDURE Main
4
5      hb_cdpSelect( "PT850" )  // Seleciona o suporte a UTF8
6
7      ? "Essa linha está dividida aqui mas " + ;
8        "será impressa apenas em uma linha"
9
10 RETURN
```

.:Resultado:.

```
Essa linha está dividida aqui mas será impressa apenas em uma linha
```

Uma string não pode ser dividida sem ser finalizada no código. Veja exemplo acima. Observe que utilizamos um operador de "+" (veremos o que é isso mais adiante) para poder concatenar a string que foi dividida.

No código a seguir temos um erro na quebra de linha pois a string não foi finalizada corretamente.

Listing 3.12: A linha não acabou (ERRO)

```
1  PROCEDURE Main
2
3      ? "Essa linha está dividida de " ; // <= Faltou o operador
4        "forma errada."
5
6      ?  "Essa linha está dividida de + ; // <= Faltou a aspa
7        forma errada."
8
9
```

```
10 RETURN
```

Outra função para o ponto e vírgula é condensar varias instruções em uma mesma linha, conforme o exemplo abaixo :

```
a := 1; ?  a; ? a + 20

equivale a

a := 1
? a
? a + 20
```

Nós desaconselhamos essa prática porque ela torna os programas difíceis de serem lidos.

Dica 6

Use o ponto e vírgula apenas para dividir uma linha grande demais.

3.7 Praticando

Procure agora praticar os seguintes casos a seguir. Em todos eles existe um erro que impede que o programa seja compilado com sucesso, fique atento as mensagens de erro do compilador e tente se familiarizar com elas.

Listing 3.13: Erro 1

```
1 /*
2 * Aprendendo Harbour
3 */
4 PROCEDURE Main
5
6     ? "Hello" "World"
7
8
9 RETURN
```

Listing 3.14: Erro 2

```
1 /*
2 * Aprendendo Harbour
3 /*
4 PROCEDURE Main
```

```
5
6       ? "Hello World"
7
8
9   RETURN
```

Listing 3.15: Erro 3

```
1   /*
2   /* Aprendendo Harbour
3   */
4   PROCEDURE Main
5
6       ? "Hello World"
7
8
9   RETURN
```

Listing 3.16: Erro 4

```
1   /*
2    Aprendendo Harbour
3   */
4   PROCEDURE Main
5
6       ? Hello World
7
8
9   RETURN
```

Listing 3.17: Erro 5

```
1   /*
2     Aprendendo Harbour
3   */
4   PROCEDURE Main
5
6       ?  "Hello ";
7           ?? "World"
8
9
10
11  RETURN
```

3.8 Desafio

Escreva um programa que coloque na tela a seguinte saída :

```
LOJAO MONTE CARLO
PEDIDO DE VENDA
========================================
ITEM            QTD      VAL         TOTAL
------------   ---- --------  ------------
CAMISA GOLA       3    10,00         30,00
LANTERNA FLA      5    50,00        250,00
                                ============
                                      280,00

 DEUS E FIEL.
```

3.9 Exercícios

1. Escreva um programa que imprima uma pergunta : "Que horas são ?" e na linha de baixo a resposta. Siga o modelo abaixo :

   ```
   Que horas são ?
   São  10:34:45.
   ```

   Dica : Use a função *TIME()* para imprimir a hora corrente.

2. Escreva um programa que apresente na tela :

   ```
   linha 1
   linha 2
   linha 3
   ```

3. Escreva um programa que exiba na tela a *string* : "Tecle algo para iniciar a 'baixa' dos documentos".

4. Escreva um programa que exiba na tela o nome do sistema operacional do computador em que ele está sendo executado. Dica: A função OS() retorna o nome do sistema operacional.

4 VARIÁVEIS E TIPOS DE DADOS

4.1 Variáveis

Variáveis são locais na memória do computador que recebem uma identificação e armazenam algum dado específico. O valor do dado pode mudar durante a execução do programa. Por exemplo, a variável **nTotalDeItens** pode receber o valor 2,4, 12, etc. Daí o nome "variável". O criador da linguagem C++ introduz assim o termo :

> basicamente, não podemos fazer nada de interessante com um computador sem armazenar dados na memória [...]. Os "lugares" nos quais armazenamos dados são chamados de *objetos*. Para acessar um objeto necessitamos de um nome. Um objeto com um nome é chamado de variável (STROUSTRUP, 2012, p. 62).

4.1.1 Criação de variáveis de memória

Uma variável deve ser definida antes de ser usada. O Harbour permite que uma variável seja definida de várias formas, a primeira delas é simplesmente criar um nome válido e definir a ele um valor, conforme a listagem 4.1. Note que a variável fica no lado esquerdo e o seu valor fica no lado direito. Entre a variável e o seu valor existe um operador (:=) que representa uma atribuição de valor. Mais adiante veremos as outras formas de atribuição, mas habitue-se a usar esse operador (Inspirado na linguagem Pascal) pois ele confere ao seu código uma clareza maior.

Listing 4.1: Criação de variáveis de memória

```
1 /*
2   Cria??o de vari?veis de mem?ria (forma 1)
3 */
4 PROCEDURE Main
5
6    cNomeQueVoceEscolher := "Vlademiro Landim Junior"
7    ? cNomeQueVoceEscolher
8
9 RETURN
```

Você não pode escolher arbitrariamente qualquer nome para nomear as suas variáveis de memória. Alguns critérios devem ser obedecidos, conforme a pequena lista logo abaixo :

1. O nome **deve** começar com uma letra ou o caracter "_" (*underscore* [1]).

[1] Em algumas publicações esse caractere recebe o nome de *underline*

2. O nome de uma variável **deve** ser formado por uma letra ou um número ou ainda por um *underscore*. Lembre-se apenas de não iniciar o nome da variável com um dígito (0...9), conforme preconiza o ítem 1.

3. Não utilize letras acentuadas ou espaços para compor o nome da sua variável.

Listing 4.2: Nomes válidos para variáveis

```
1  /*
2    Nomes v?lidos
3  */
4  PROCEDURE MAIN()
5
6     nOpc := 100
7     _iK := 200
8     nTotal32 := nOpc + _iK
9
10 RETURN
```

É aconselhável que uma variável NÃO tenha o mesmo nome de uma palavra reservada da linguagem Harbour. Palavras reservadas são aquelas que pertencem a própria sintaxe da linguagem, como TOTAL, USE, REPLACE, etc. A lista de todas as palavras reservadas encontra-se no Apêndice X, mas você não deve se preocupar em decorar todas as palavras reservadas. O que você precisa é seguir algumas regras simples para se nomear uma variável, conforme veremos a seguir.

Dica 7

Em determinadas linguagens, como a Linguagem C, você não pode nomear uma variável com o mesmo nome de uma palavra reservada. O Clipper (ancestral do Harbour) também não aceita essa prática, de acordo com (RAMALHO, 1991, p. 68). O Harbour [a] não reclama do uso de palavras reservadas, mas reforçamos que você **não** deve adotar essa prática.

Listing 4.3: Uso de palavras reservadas

```
1  /*
2    Evite usar palavras reservadas
3  */
4  PROCEDURE Main
5
6     PRIVATE := 100 // PRIVATE ? uma palavra reservada
7     REPLACE := 12  // REPLACE tamb?m ?
8     TOTAL := PRIVATE + REPLACE // Total tamb?m ?
9     ? TOTAL
10
11 RETURN
```

.:Resultado:.

```
112
```

[a] A versão do Harbour que nós usamos para desenvolver esse documento foi o 3.2.0

Dica 8

(SPENCE, 1994, p. 11) nos informa que o Clipper possui um arquivo (chamado "reserved.ch") na sua pasta include com a lista de palavras reservadas. O Harbour também possui esse arquivo, mas a lista está em branco. Você pode criar a sua própria lista de palavras reservadas no Harbour. Caso deseje fazer isso você precisa seguir esses passos :

1. Edite o arquivo reserved.ch que vem com o Harbour na sua pasta include.
2. Inclua em uma lista (um por linha) as palavras que você quer que sejam reservadas.
3. No seu arquivo de código (.prg) você deve incluir (antes de qualquer função) a linha : *#include "reserved.ch"*.

O Harbour irá verificar, em tempo de compilação, a existência de palavras que estejam nesse arquivo e irá barrar a compilação caso o seu código tenha essas palavras reservadas (por você). Você pode, inclusive, criar a sua própria lista de palavras reservadas independente de serem comandos ou não.

O programa a seguir (Listagem 4.4) exemplifica uma atribuição de nomes inválidos à uma variável de memória.

Listing 4.4: Nomes inválidos para variáveis

```
1 /*
2   Nomes inv?lidos
3 */
4 PROCEDURE Main
5
6    90pc := 100 // Inicializa com um n?mero
7
8 RETURN
```

4.1.2 Seja claro ao nomear suas variáveis

Tenha cuidado quando for nomear suas variáveis. Utilize nomes indicativos daquilo que elas armazenam. Como uma variável não pode conter espaços em branco, utilize caracteres *underscore* para separar os nomes, ou então utilize uma combinação de letras maiúsculas e

minúsculas [2].

Listing 4.5: Notações

```
/*
Notações utilizadas
*/
PROCEDURE Main

   Tot_Nota := 1200 // Variável numérica (Notação com underscores)
                    // que representa o total da nota fiscal

   NomCli := "Rob Pike" // Variável caracter (Notação hungara)
                        // que representa o nome do cliente


RETURN
```

Mais adiante estudaremos os tipos de dados, mas já vamos adiantando : procure prefixar o nome de suas variáveis com o tipo de dado a que ela se refere. Por exemplo: **nTot** [3] representa uma variável numérica, por isso ela foi iniciada com a letra "n". Já **cNomCli** representa uma variável caractere (usada para armazenar *strings*), por isso ela foi iniciada com a letra "c". Mais adiante estudaremos com detalhes outros tipos de variáveis e aconselhamos que você siga essa nomenclatura: "n" para variáveis numéricas, "c" para variáveis caracteres (strings), "d" para variáveis do tipo data, "l" para variáveis do tipo lógico, etc.

Dica 9

Essa parte é tão importante que vamos repetir : "apesar de ser simples criar um nome para uma variável, você deve priorizar a clareza do seu código. O nome de uma variável deve ser informativo, conciso, memorizável e, se possível, pronunciável" (KERNIGHAN; PIKE, 2000, p. 3). Procure usar nomes descritivos, além disso, procure manter esses nomes curtos, conforme a listagem 4.6.

Listing 4.6: Nomes curtos e concisos para variáveis

```
/*
  NOMES CONSISOS

  Adaptado de (KERNIGHAN, p.3, 2000)
*/
PROCEDURE Main

   /*
   Use comentários "//" para acrescentar informações sobre
```

[2] Essa notação é conhecida como Notação Hungara.

[3] Essa notação é chamada por Rick Spence de "Notação Hungara Modificada"

```
10      a variável, em vez de deixar o nome da variável grande
            demais.
11      */
12      nElem := 1 // Elemento corrente
13      nTotElem := 1200 // Total de elementos
14
15  RETURN
```

Evite detalhar demais conforme a listagem 4.7.

Listing 4.7: Nomes longos e detalhados demais

```
1  /*
2    NOMES GRANDES DEMAIS PARA VARI?VEIS. EVITE ISSO!!!
3
4    Adaptado de (KERNIGHAN, p.3, 2000)
5  */
6  PROCEDURE Main
7
8     nNumeroDoElementoCorrente := 1
9     nNumeroTotalDeElementos := 1200
10
11 RETURN
```

4.1.3 Atribuição

Já vimos como atribuir dados a uma variável através do operador *:=* que foi inspirado na Linguagem Pascal. Agora iremos detalhar essa forma e ver outras formas de atribuição.

O código listado em 4.8 nos mostra a atribuição com o operador ‘‘=’’ e com o comando STORE. Conforme já dissemos, prefira o operador *:=*.

Listing 4.8: Outras forma de atribuição

```
1  /*
2  Atribui??o
3  */
4  REQUEST HB_CODEPAGE_PT850
5
6  PROCEDURE Main
7
8   hb_cdpSelect( "PT850" )
9
10    x = 1200  // Atribu? 1200 ao valor x
11    STORE 100 TO y // Atribu? 100 ao valor y
12
13    ? x + y
```

```
14
15 RETURN
```

.:Resultado:.

```
      1300
```

A atribuição de variáveis também pode ser feita de forma simultânea, conforme a listagem 4.9.

Listing 4.9: Atribuição simultânea

```
1  /*
2  Atribui??o
3  */
4  REQUEST HB_CODEPAGE_PT850
5
6  PROCEDURE Main
7
8   hb_cdpSelect( "PT850" )
9
10 LOCAL x := y := z := k := 100
11
12    ? x + y + z + k
13
14 RETURN
```

Dica 10

Nos capítulos seguintes nós iremos nos aprofundar no estudo das variáveis, mas desde já é importante que você adquira bons hábitos de programação. Por isso iremos trabalhar com um tipo de variável chamada de *LOCAL*. Você não precisa se preocupar em entender o que esse tipo significa, mas iremos nos acostumar a declarar as variáveis como *LOCAL* e **obrigatoriamente** no início do bloco de código, assim como feito no código anterior (Listagem 4.9). Note também que a palavra reservada *LOCAL* não recebe indentação, pois nós a consideramos parte do início do bloco e também que nós colocamos uma linha em branco após a sua declaração para destacá-las do restante do código. Nós também colocamos um espaço em branco após a vírgula que separa os nomes das variáveis, pois isso ajuda na visualização do código.
A existência de linhas e espaços em branco não deixa o seu programa "maior" e nem consome memória porque o compilador simplesmente os ignora. Assim, você deve usar os espaços e as linhas em branco para dividir o seu programa em "mini-pedaços" facilmente visualizáveis.
Podemos sintetizar o que foi dito através da seguinte "equação" :

```
Variáveis agrupadas de acordo com o comentário +
Indentação no início do bloco +
Espaço em branco após as vírgulas +
Linhas em branco dividindo os ‘‘mini-pedaços’’  =
--------------------------------------------------
Código mais fácil de se entender.
```

Dica 11

Já foi dito que o Harbour é uma linguagem “Case insensitive”, ou seja, ela não faz distinção entre letras maiúsculas e minúsculas. Essa característica se aplica também as variáveis. Os seguintes nomes são equivalentes : **nNota, NNOTA, nnota, NnOta e NNota**. Note que algumas variações da variável **nNota** são difíceis de se ler, portanto você deve sempre nomear as suas variáveis obedecendo aos padrões já enfatizados nesse capítulo.

4.2 Recebendo dados do usuário

No final desse capítulo iremos treinar algumas rotinas básicas com as variáveis que aprendemos, mas boa parte dessas rotinas precisam que você saiba receber dados digitados pelo usuário. O Harbour possui várias maneiras de receber dados digitados pelo usuário, como ainda estamos iniciando iremos aprender uma maneira simples de receber dados numéricos: o comando *INPUT*[4]. O seu uso está ilustrado no código 4.10.

Listing 4.10: Recebendo dados externos digitados pelo usuário

```
1  /*
2  Uso do comando INPUT
3  */
4  REQUEST HB_CODEPAGE_PT850
5  PROCEDURE Main
6  LOCAL nVal1 := 0 // Declara a variável parcela de pagamento
7  LOCAL nVal2 := 0 // Declara a variável parcela de pagamento
8
9     hb_cdpSelect( "PT850" ) // Seleciona o suporte a UTF8
10
11    INPUT "Informe o primeiro valor : " TO nVal1
12    INPUT "Informa o segundo valor : " TO nVal2
13
14    ? "A soma dos dois valores é : ", nVal1 + nVal2
15
16
17 RETURN
```

[4]O comando INPUT recebe também dados caracteres, mas aconselhamos o seu uso para receber apenas dados numéricos.

Após digitar o número tecle *ENTER*.

Dica 12

Note que, na listagem 4.10 as variáveis foram declaradas em uma linha separada, enquanto que na listagem 4.9 elas foram declaradas em uma única linha. Não existe uma regra fixa com relação a isso, mas aconselhamos você a organizar as variáveis de acordo com o comentário (*//*) que provavelmente você fará após a declaração. Por exemplo, se existem duas variáveis que representam coisas que podem ser comentadas em conjunto, então elas devem ser declaradas em apenas uma linha. A listagem 4.10 ficaria mais clara se as duas linhas de declaração fossem aglutinadas em apenas uma linha. Isso porque as duas variáveis (*nVal1* e *nVal2*) possuem uma forte relação entre si. O ideal seria :

```
LOCAL nVal1 := 0, nVal2 := 0 // Parcelas do pagamento.
```

ou ainda recorrendo a atribuição simultânea de valor :

```
LOCAL nVal1 :=  nVal2 := 0 // Parcelas do pagamento.
```

Note também que nós devemos atribuir um valor as variáveis para informar o seu tipo numérico (no caso do exemplo acima, o zero informa que as variáveis devem ser tratadas como numéricas).

Se os dados digitados forem do tipo caractere você deve usar o comando *ACCEPT*. (Veja um exemplo na listagem 4.11). Observe que o *ACCEPT* funciona da mesma forma que o *INPUT*.

Listing 4.11: Recebendo dados externos digitados pelo usuário (Tipo caractere)

```
1  /*
2  Uso do comando ACCEPT
3  */
4  REQUEST HB_CODEPAGE_PT850
5  PROCEDURE Main
6  LOCAL cNome := "" // Seu nome
7
8     HB_CdpSelect( "PT850" )
9
10    /* Pede e exibe o nome do usuário */
11    ACCEPT "Informe o seu nome : " TO cNome
12    ? "O seu nome é : ", cNome
13
```

```
14 RETURN
```

Após digitar a *string* tecle *ENTER*.

Dica 13

Você pode questionar porque temos um comando para receber do usuário dados numéricos e outro para receber dados caracteres. Esses questionamentos são pertinentes, mas não se preocupe pois eles (*ACCEPT* e *INPUT*) serão usados apenas no início do nosso aprendizado. Existem formas mais eficientes para a entrada de dados, mas o seu aprendizado iria tornar o processo inicial de aprendizado da linguagem mais demorado.
Apenas aprenda o "mantra" abaixo e você não terá problemas :

```
Para receber dados do tipo numérico = use INPUT.
Para receber dados do tipo caractere = use ACCEPT.
```

Se mesmo assim você quer saber o porque dessa diferença continue lendo, senão pode pular o trecho a seguir e ir para a seção de exemplos.
Na primeira metade da década de 1980 o dBase II foi lançado. Naquela época não tínhamos os recursos de hardware e de software que temos hoje, tudo era muito simples. Os primeiros comandos de entrada de dados usados pelo dBase foram o *INPUT* e o *ACCEPT*. O *ACCEPT* serve para receber apenas variáveis do tipo caracter. Você não precisa nem declarar a variável, pois se ela não existir o comando irá criá-la para você. O *INPUT* funciona da mesma forma, mas serve também para receber dados numéricos, caracteres, data e lógico. Porém você deve formatar a entrada de dados convenientemente, e é isso torna o seu uso confuso para outros tipos de dados que não sejam os numéricos.
Exemplos de uso do comando INPUT :

```
Usando com variáveis numéricas :

    nVal := 0
    INPUT ‘‘Digite o valor : ‘‘ TO nVal

Usando com variáveis caracteres :

    cNome := ‘‘ ’’
    INPUT ‘‘Digite o seu nome : ‘‘ TO ‘‘cNome’’
    // Note que  variável cNome deve estar entre parênteses
```

Vidal acrescenta que "o comando *INPUT* deve preferivelmente ser utilizado para a entrada de dados numéricos. Para a entrada de dados caracteres use o comando *ACCEPT*"

(VIDAL, 1989, p. 107). Para usar *INPUT* com dados do tipo data utilize a função CTOD() e com dados do tipo lógico apenas use o dado diretamente (.t. ou .f.)[a].

[a]Veremos os dados lógico e data mais adiante.

4.3 Exemplos

4.3.1 Realizando as quatro operações

O exemplo da listagem 4.12 gera um pequeno programa que executa as quatro operações com dois números que o usuário deve digitar. Note que o sinal de multiplicação é um asterisco (“*”) e o sinal de divisão é uma barra utilizada na escrita de datas (“/”) [5].

Listing 4.12: As quatro operações

```
1  /*
2  As quatro operações
3  */
4  REQUEST HB_CODEPAGE_PT850
5
6  PROCEDURE Main
7  LOCAL nValor1 := nValor2 := 0 // Valores a serem calculados
8
9     hb_cdpSelect( "PT850" ) // Seleciona o suporte a UTF8
10
11    // Recebendo os dados
12    ? "Introduza dois números para que eu realize as quatro oper.: "
13    INPUT "Introduza o primeiro valor : " TO nValor1
14    INPUT "Introduza o segundo valor : " TO nValor2
15
16    // Calculando e exibindo
17    ? "Adição.......... : " , nValor1 + nValor2
18    ? "Subtração....... : " , nValor1 - nValor2
19    ? "Multiplicação.... : " , nValor1 * nValor2
20    ? "Divisão......... : " , nValor1 / nValor2
21
22
23 RETURN
```

[5]O sinal de uma operação entre variáveis recebe o nome de “operador”. Veremos mais sobre os operadores nos capítulos seguintes.

4.3.2 Calculando o antecessor e o sucessor de um número

O exemplo da listagem 4.12 gera um pequeno programa que descobre qual é o antecessor e o sucessor de um número qualquer inserido pelo usuário.

Listing 4.13: Descobrindo o antecessor e o sucessor

```
1  /*
2  Descobre o  antecessor e o sucessor
3  */
4  REQUEST HB_CODEPAGE_PT850
5
6  PROCEDURE Main
7  LOCAL nValor := 0 // Número a ser inserido
8
9     hb_cdpSelect( "PT850" ) // Seleciona o suporte a UTF8
10
11    // Recebendo os dados
12    ? ""
13    ? "**** Descobrindo o antecessor e o sucessor ****"
14    ? ""
15    INPUT "Introduza o número : " TO nValor
16
17    // Calculando e exibindo
18    ? "Antecessor........ : " , nValor - 1
19    ? "Sucessor.......... : " , nValor + 1
20
21 RETURN
```

4.4 Exercícios

1. (MIZRAHI, 1990, p. 25) - Escreva um programa que declare 3 variáveis e atribua os valores 1,2 e 3 a elas; depois mais 3 variáveis e atribua a elas as letras a,b e c; finalmente imprima na tela :

   As variáveis inteiras contem os números 1, 2 e 3. As variáveis caracteres contem os valores a, b e c.

2. (HORSTMAN, 2005, p. 47) Escreva um programa que exibe a mensagem “Oi, meu nome é Hal!”. Então, em uma nova linha, o programa deve imprimir a mensagem “O que você gostaria que eu fizesse ?”. Então é a vez do usuário digitar uma entrada. [...] Finalmente, o programa deve ignorar a entrada do usuário e imprimir uma mensagem “Sinto muito, eu não posso fazer isto.”. Aqui está uma execução típica :

```
Oi, meu nome é Hal!
O que você gostaria que eu fizesse ?
A limpeza do meu quarto.
Sinto muito, eu não posso fazer isto.
```

3. (HORSTMAN, 2005, p. 47) Escreva um programa que imprima uma mensagem "Oi, meu nome é Hal!". Então, em uma nova linha, o programa deve imprimir a mensagem "Qual o seu nome ?" [...] Finalmente, o programa deve imprimir a mensagem "Oi, *nome do usuário*. Prazer em conhecê-lo!" Aqui está uma execução típica :

```
Oi, meu nome é Hal!
Qual é o seu nome ?
Dave
Oi, Dave. Prazer em conhecê-lo.
```

4. Escreva um programa que receba do usuário o nome e a idade dele . Depois de receber esses dados o programa deve exibir o nome e a idade do usuário convertida em meses. Use o exemplo abaixo como modelo :

```
Digite o seu nome : Paulo
Digite quantos anos você tem : 20

Seu nome é Paulo e você tem aproximadamente 240 meses de vida.
```

5. Escreva um programa que receba do usuário um valor em horas e exiba esse valor convertido em segundos. Conforme o exemplo abaixo :

```
Digite um valor em horas : 3
3    horas tem       10800      segundos
```

4.5 Desafios

4.5.1 Identifique o erro de compilação no programa abaixo.

Listing 4.14: Erro 1

```
/*
Onde está o erro ?
*/
PROCEDURE Main
LOCAL x, y, x // Número a ser inserido

   x := 5
   y := 10
   x := 20

RETURN
```

4.5.2 Identifique o erro de lógica no programa abaixo.

Um erro de lógica é quando o programa consegue ser compilado mas ele não funciona como o esperado. Esse tipo de erro é muito perigoso pois ele não impede que o programa seja gerado. Dessa forma, os erros de lógica só podem ser descoberto durante a seção de testes ou (pior ainda) pelo cliente durante a execução. Um erro de lógica quase sempre é chamado de *bug*.

Listing 4.15: Erro de lógica

```
/*
Onde está o erro ?
*/
PROCEDURE Main
LOCAL x, y // Número a ser inserido

   x := 5
   y := 10
   ACCEPT "Informe o primeiro número : " TO x
   ACCEPT "Informe o segundo número : " TO y
   ? "A soma  é ", x + y

RETURN
```

4.5.3 Valor total das moedas

(HORSTMAN, 2005, p. 50) Eu tenho 8 moedas de 1 centavo, 4 de 10 centavos e 3 de 25 centavos em minha carteira. Qual o valor total de moedas ? Faça um programa que calcule o valor total para qualquer quantidade de moedas informadas.

Siga o modelo :

```
Informe quantas moedas você tem :

Quantidade de moedas de 1 centavo :  10
Quantidade de moedas de 10 centavos : 3
Quantidade de moedas de 25 centavos : 4

Você tem     1.40 em moedas.
```

REFERÊNCIAS BIBLIOGRÁFICAS

AGUILAR, L. J. **Fundamentos de programação: algoritmos, estruturas de dados e objetos**. Editora McGraw-Hill, São Paulo, 2008.

ASCENCIO, A. F. G.; CAMPOS, E. A. V. de. **Fundamentos da programação de computadores: algoritmos, PASCAL, C/C++ (padrão ANSI) e JAVA**. Pearson, São Paulo, 2014.

DAMAS, L. **Linguagem C**. LTC, Rio de Janeiro, 2013.

DEITEL, H. M.; DEITEL, P. J. **C++ Como programar**. Bookman, Porto Alegre, 2001.

FORBELLONE, A. L.; EBERSPACHER, H. F. **Lógica de programação: a construção de algoritmos e estrutura de dados**. Prentice Hall, São Paulo, 2005.

GUIMARãES, n. d. M.; LAGES, N. A. d. C. **Algoritmos e estruturas de dados**. LTC - Livros Técnicos e Científicos, Rio de Janeiro, 1994.

HORSTMAN, C. **Conceitos de computação com o essencial de C++ .** Bookman, Porto Alegre, 2005.

HYMAN, M. I. **Borland C++ para leigos**. Berkeley Brasil Editora, São Paulo, 1995.

KERNIGHAN, B. W.; PIKE, R. **A prática da programação**. Campus, Rio de Janeiro, 2000.

KERNIGHAN, B. W.; RITCHIE, D. M. **C: a linguagem de programação**. Campus, Rio de Janeiro, 1986.

KRESIN, A. *Harbour for beginners*. 2016. `http://www.kresin.ru/en/hrbfaq.html/`. [Online; accessed 13-Ago-2016].

MIZRAHI, V. V. **Treinamento em linguagem C.** McGraw-Hill, São Paulo, 1990.

MIZRAHI, V. V. **Treinamento em linguagem C++.** Pearson, São Paulo, 2004.

NANTUCKET. **Clipper 5.0 - manual de referência**. Nantucket Corporation, 1990.

PAPAS, C. H.; MURRAY, W. H. **Borland C++ 4**. Makron, Rio de Janeiro, 1994.

RAMALHO, J. A. A. **Clipper 5.0 - Básico**. McGraw-Hill, São paulo, 1991.

SPENCE, R. **Clipper 5.0 - release 5.01**. Makron, Rio de Janeiro, 1991.

SPENCE, R. **Clipper 5.2**. Makron, Rio de Janeiro, 1994.

STROUSTRUP, B. **Princípios e práticas de programação com C++ .** Bookman, Porto Alegre, 2012.

SWAN, T. **Tecle e aprenda C**. Berkeley Brasil editora, São Paulo, 1994.

TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. **Estrutura de dados usando C.** Makron Books, São Paulo, 1995.

VIDAL, A. G. d. R. **Clipper**. LTC - Livros Técnicos e Científicos, Rio de Janeiro, 1989.

VIDAL, A. G. d. R. **Clipper 5**. LTC - Livros Técnicos e Científicos, Rio de Janeiro, 1991.

YOURDON, E. **Análise estruturada moderna**. Campus, Rio de Janeiro, 1992.